

# Volume Cost Based Mesh Simplification

Chansophea Chuon

*Computer Science and Information Management  
Asian Institute of Technology  
Klong Luang, Pathumthani 12120, THAILAND  
chansophea.chuon@ait.asia*

Sumanta Guha

*Computer Science and Information Management  
Asian Institute of Technology  
Klong Luang, Pathumthani 12120, THAILAND  
guha@ait.asia*

**Abstract**—We develop a polygonal mesh simplification algorithm using a vertex-decimation approach. The novelty in our method lies in (a) a characterization of mesh vertices as hyperbolic or non-hyperbolic based upon their discrete local geometry, (b) the cost function used to select a vertex for decimation, and (c) the heuristics applied to re-triangulate the resulting hole. The algorithm begins by classifying the input mesh vertices as hyperbolic or non-hyperbolic, and then computes a volume cost for each non-hyperbolic vertex, in analogy with spherical volume, to capture the loss of fidelity if that vertex is decimated. Vertices of least volume cost are successively deleted and the resulting hole re-triangulated. Preliminary experiments indicate a performance comparable to that of the best known mesh simplification algorithms.

**Keywords**-Hyperbolic vertex; level of detail; local geometry; mesh simplification; multi-resolution; vertex decimation; volume cost.

## I. INTRODUCTION

Polygonal mesh models are ubiquitous in computer graphics. They are typically either user-made in a 3D design environment, or automatically generated by triangulating point cloud data obtained from a digital scan of some real-life object. Polygonal models are widely supported in both design and rendering, there being available plentiful high-quality commercial and free software.

The visual quality of a model depends on the number of polygons (the more usually the better) and the quality of the polygons themselves (the fewer slivers the better). However, a large polygon count while making for a visually attractive surface taxes the resources of the rendering device. Particularly, in real-time and interactive applications too large a polygon count may cause unacceptable degradation of performance. Applications that require mesh models to be transferred over a network are sensitive to polygon count as well.

For these reasons there has been considerable interest over more than a decade in mesh simplification algorithms. The goal is to accept as input a complex model and simplify it to various levels of resolution by reducing the number of polygons, at the same time retaining as far as possible fidelity to the original. Figure 1 shows a cow at three different levels of resolution (the simplification was done with our new software).

In this paper we propose a new mesh simplification algorithm. We assume as input a triangulated mesh (polygonal meshes are straightforwardly triangulated). Our algorithm itself falls within the category of simplification algorithms based on vertex decimation. A vertex decimation method typically processes the input mesh by iteratively deleting a vertex and its adjacent faces and then re-triangulating the resulting hole. A candidate vertex for decimation is commonly chosen by minimizing the value of some cost function over the current vertex set.

The novelty in our algorithm lies firstly in a new scheme to classify mesh vertices as hyperbolic and non-hyperbolic. This scheme is local and discrete, and not derived from the well-known method of differentiating between hyperbolic, parabolic and elliptic points on a smooth surface using Gaussian curvature, though the two are not unrelated. (Non-hyperbolic vertices are further classified as convex or concave, though we do not apply the distinction in our algorithm itself.) This discrete characterization of vertices seems to be new in CG applications, and more intrinsically suited to meshed surfaces than existing methods that distinguish vertices by estimating pseudo-curvature values on a smooth approximation.

We apply our characterization to select non-hyperbolic vertices, each of which is then associated with a measure of the volume that it covers – this so-called volume cost is determined by the neighborhood geometry of the vertex. Hyperbolic vertices, on the other hand, do not in any natural manner cover a volume, and are always preserved through our algorithm. Non-hyperbolic vertices are decimated in order of least volume cost. A new re-triangulation scheme is proposed as well to patch the holes arising from vertex decimation.

We have implemented our algorithm and preliminary experiments suggest a quality of simplification comparable to current best-known algorithms. Our software will be made available freely.

The rest of the paper is organized as follows. In Section II we briefly discuss the various approaches currently to mesh simplification. Its underlying theory and our simplification algorithm are described in Section III. Experimental results are discussed and output shown in Section IV. We conclude

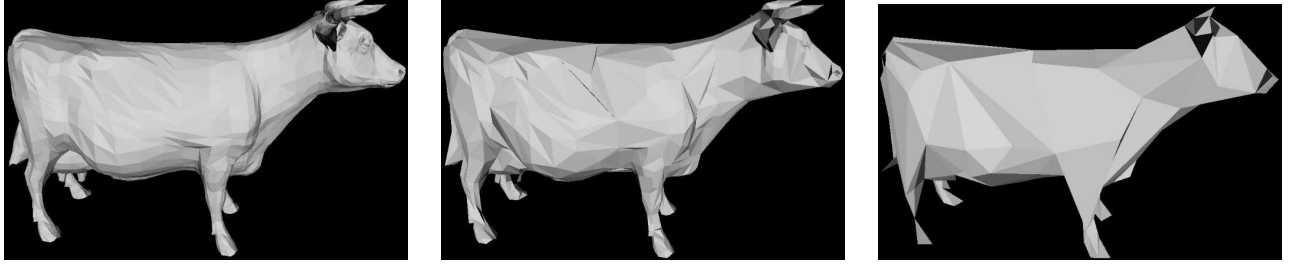


Figure 1. Cow at 3 different resolutions: 5804, 1772 and 328 triangles.

in Section V.

## II. BACKGROUND AND RELATED WORK

The problem of mesh simplification has been of interest in the rendering community for several years. Various approaches have been developed and implemented. Surveys of these include ones by Cignoni et al [3], Garland [4], Luebke [8], [9] and Talton [15]. The best-known approaches to mesh simplification can be broadly classified into three categories – vertex clustering, edge contraction and vertex decimation – of which ours falls into the latter.

The vertex clustering method, introduced by Rossignac and Borrel [12], places a bounding box around the model, which is then divided into a uniform 3D grid of rectilinear cells. Vertices inside each cell are merged into a single representative vertex (Figure 2). The representative vertex is chosen either from amongst the original ones in the cell – based on a heuristic comparison of their contribution to the appearance of the surface in that cell – or as a weighted average.

The vertex clustering method is simple, intuitive and easy to implement. Moreover, it lends itself to graceful progressive simplification by means of a gradual increase in cell size. However, the fidelity to the original resulting from vertex clustering is often poor, particularly as the error bounds are not determined intrinsically by the model, but by the size of the grid cells.

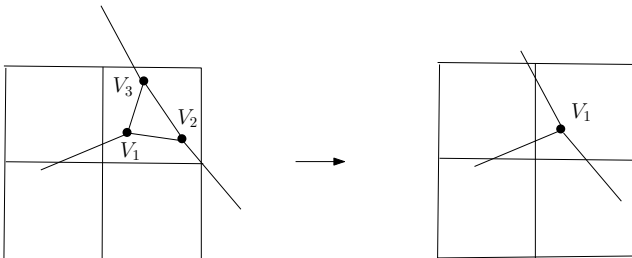


Figure 2. Vertex clustering.

In edge contraction, typically, edges are contracted iteratively into a single vertex each, and adjacent faces updated accordingly (Figure 3). The difference between edge contracting algorithms lies primarily in the cost function

which each tries to minimize when choosing a candidate to contract.

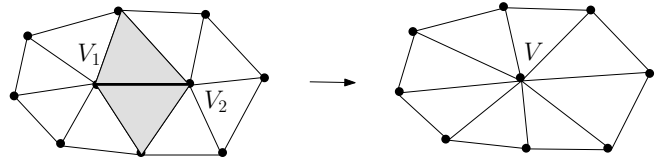


Figure 3. Edge contraction.

The most efficient edge contraction algorithm to date seems to be the one by Garland and Heckbert [5], [6], which applies a quadric error cost function. This measures the sum of the squares of the distances of the vertex to which an edge is collapsed from the faces adjacent to the latter, a measure of the loss of quality from that particular collapse. Quadric-based mesh simplification is efficient and produces provably high-quality output. Hoppe [7] proposes a notable application of the edge contraction approach to produce a continuous range of resolution levels via the so-called progressive mesh representation.

Schroeder et al [13] introduce the vertex decimation technique, where vertices are successively removed, each together with its adjacent faces (Figure 4). As in edge collapsing, a cost function is applied to determine the candidate to remove. The original cost function proposed by Schroeder et al is either the distance of the vertex to the best-fit plane of its neighbors, or that to a boundary edge, depending on the type of vertex.

The second step of a vertex decimation method, following the removal of a vertex and its adjacent faces, is closing the resulting hole, which is bounded by a loop, by means of re-triangulation. Schroeder et al apply a recursive procedure, repeatedly splitting the loop into two by means of a diagonal chosen to maximize the aspect ratio of the resulting two sub-loops.

Though there seems to exist little in the literature by way of theoretical proofs, vertex decimation methods – many variations of Schroeder’s original have since been proposed and implemented – seem to work well in practice.

Our method is based on vertex decimation as well. We begin, however, with a novel classification of vertices into

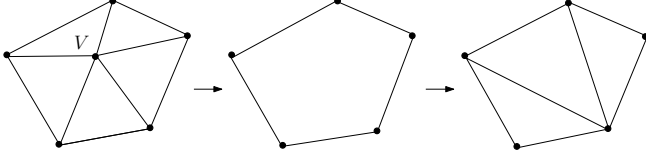


Figure 4. Vertex decimation.

hyperbolic and non-hyperbolic, and then restrict our search for a candidate to decimate amongst non-hyperbolic vertices. This allows application of a volume-based cost function which is geometrically meaningful for non-hyperbolic vertices. Minimizing volume loss at each step ensures shape fidelity. We do not as yet have any proofs of quality, but experimentation with numerous surface meshes suggests that our method is competitive with those currently popular in practice.

### III. THEORY AND ALGORITHM

#### A. Vertex Classification

To simplify the theoretical development we assume that the input mesh is a topologically closed surface, implying that it has a meaningful interior. This assumption is not essential, however, and our implementation works for non-closed meshes as well. We assume as well that each mesh face is triangular. We begin with a characterization of vertices as convex, concave or hyperbolic.

**Definition 1.** A vertex  $V$  of a mesh  $M$  is hyperbolic if it is contained in the interior (as a subspace of  $\mathbb{R}^3$ ) of the convex hull  $C$  of its neighbors. A vertex  $V$  that is not hyperbolic is convex if there exists a (flat) disc  $D$  centered at  $V$  which does not intersect the interior of  $M$ ; otherwise, it is concave.

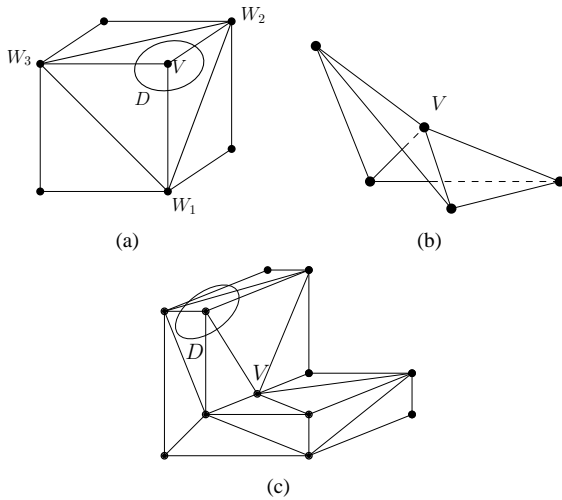


Figure 5. Illustrations of convex, hyperbolic and concave vertices.

For example, the convex hull of the neighbors of  $V$  in Figure 5(a) is the triangle  $W_1W_2W_3$ , which has empty interior

in  $\mathbb{R}^3$ . Therefore,  $V$  is trivially non-hyperbolic; moreover, the disc  $D$  proves that  $V$  is convex. The vertex  $V$  on the saddle-shaped surface in Figure 5(b) is hyperbolic. The reader may check in Figure 5(c) that all the vertices on the L-shaped solid are non-hyperbolic, and that only vertex  $V$  is concave, while all the rest are convex. The disc  $D$  at a corner of the L-shaped solid indicates the reason why we cannot replace the disc in the definition of convexity with its containing plane – a plane may intersect the mesh at a distant point.

Our characterization of a mesh vertex is not unrelated to that of a point of a smooth surface as hyperbolic, parabolic or elliptic by means of Gaussian curvature, but we'll not explore the connection here. It should be observed though that use of discrete local geometry to classify vertices seems more natural for a meshed surface than computing pseudo-curvature values, as some authors do, on a smooth approximation.

Whether a vertex  $V$  is hyperbolic or not is a local decision depending on its disposition with respect to its neighbors. However, distinguishing a non-hyperbolic vertex  $V$  as either convex or concave necessitates one to determine the side of the surface near  $V$  that the interior of  $M$  lies, which requires global knowledge of  $M$ .

Our approach though requires only to be able to distinguish if a vertex is hyperbolic or non-hyperbolic, and not further between convex and concave. Therefore, we need know only of the local geometry at each vertex.

#### B. Volume Cost Estimation

The link of a vertex  $V$  of a mesh  $M$  is the union of its neighboring vertices and edges between them. E.g., the link of  $V$  in Figure 5(a) is the boundary of the triangle  $W_1W_2W_3$ .

We would like to measure the solid angle subtended at a convex or concave vertex by its link. We'll give a constructive definition of this angle. But, first, we need a classical formula for the solid angle subtended by a triangle at a point.

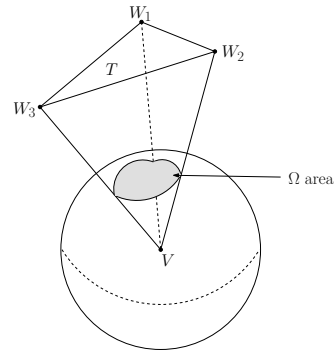


Figure 6. Solid angle.

The solid angle  $\Omega$  subtended by a triangle  $T$  at a point  $V$  not belonging to  $T$  is the surface area of the projection of

$T$  onto the unit sphere centered at  $V$  (the unit of solid angle is *steradians* but we'll not use this term). See Figure 6. Van Oosterom and Strackee [11] give the following formula if the corners of  $T$  are  $W_1, W_2$  and  $W_3$ :

$$\tan\left(\frac{\Omega}{2}\right) = \frac{[\vec{W}_1 \vec{W}_2 \vec{W}_3]}{(\vec{W}_1 \cdot \vec{W}_2)|\vec{W}_3| + (\vec{W}_1 \cdot \vec{W}_3)|\vec{W}_2| + (\vec{W}_2 \cdot \vec{W}_3)|\vec{W}_1| + |\vec{W}_1||\vec{W}_2||\vec{W}_3|}$$

where  $\vec{W}_i$  is the position vector of  $W_i$  w.r.t.  $V$  and  $[\vec{W}_1 \vec{W}_2 \vec{W}_3]$  denotes a scalar triple product.

The preceding formula leads to the following constructive definition of what we call, simply, the *angle* at a non-hyperbolic mesh vertex  $V$  and denote by  $angle(V)$ .

**Definition 2.** Suppose that  $V$  is a non-hyperbolic vertex  $V$  of a mesh  $M$ . If  $V$  lies on the convex hull  $C$  of its neighbors, then set  $angle(V)$  to  $2\pi$ . Otherwise, let  $P$  be a plane that separates  $V$  from  $C$ , not intersecting either. Suppose the neighbors of  $V$  in cyclic order are  $W_1, W_2, \dots, W_k$ . Let the straight line through  $V$  and  $W_i$  intersect  $P$  at  $W'_i$ . Triangulate the (plane, simple) polygon  $W'_1 W'_2 \dots W'_k$ . Set  $angle(V)$  to be the sum of the solid angles subtended at  $V$  by the triangles in this triangulation (Figure 7).

*Note 1.* The case that  $V$  lies on the convex hull  $C$  of its neighbors arises precisely when these neighboring vertices all lie on a plane that contains  $V$  as well (e.g., imagine an additional vertex in the middle of a face of the cube in Figure 5(a)).

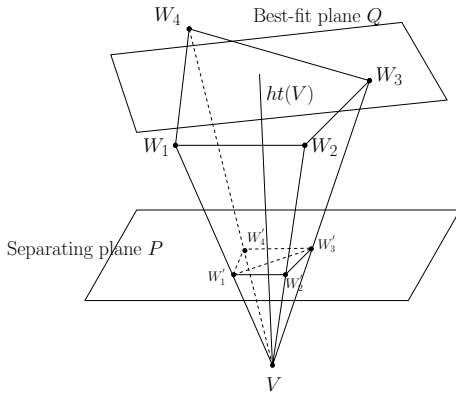


Figure 7. Using a separating plane to calculate  $angle(V)$  and the best-fit plane to calculate  $ht(V)$ .

Next, we need a measure of the distance of a non-hyperbolic vertex  $V$  from its neighbors, which are, say,  $W_1, W_2, \dots, W_k$ . Accordingly:

**Definition 3.** For a non-hyperbolic vertex  $V$  define its height  $ht(V)$  to be the distance of  $V$  to the best-fit plane  $Q$  of  $W_1, W_2, \dots, W_k$  (Figure 7).

*Note 2.* Neither  $angle(V)$  nor  $ht(V)$  is meaningful for a hyperbolic vertex as then  $V$ , in a sense, is “enclosed” by its neighbors.

Finally, we would like to estimate the volume “covered” by a non-hyperbolic  $V$  and its adjacent faces. The reason is that this volume will provide an estimate of the cost of decimating  $V$ : intuitively, the volume covered by  $V$  is lost if  $V$  is convex, or added if  $V$  is concave. Unfortunately, there is generally no well-defined closed space covered by  $V$  and its adjacent faces, as this requires “filling” the link of  $V$  with a 2D surface, which is not possible in an unambiguous manner if the link is non-planar (e.g., consider if  $W_1, W_2, W_3$  and  $W_4$  are non-coplanar in Figure 7). However, the following formula, based on computing a spherical volume, seems a reasonable estimate in most well-behaved situations.

**Definition 4.** For a non-hyperbolic vertex  $V$  define its volume  $vol(V)$  by

$$vol(V) = angle(V) \times ht(V)^3$$

The formula is motivated by the expression  $\frac{1}{3}\Omega r^3$  for the volume of a sector of a sphere, where  $\Omega$  is the solid angle subtended at the center by the sector and  $r$  is the radius of the sphere.

### C. Simplification Algorithm

Our simplification algorithm is now straightforward to describe. Firstly, we compute  $angle(V)$ ,  $ht(V)$  and then  $vol(V)$  for each non-hyperbolic vertex of the input mesh  $M$ . The value  $vol(V)$  is assigned as the cost of decimating  $V$ . However, prior to using this cost function, a simple heuristic is implemented to preserve sharp-angled features whose volume might be small (e.g., Figure 8) as follows: we restrict the competition to find the least-cost vertex to those whose angles are at least as large as the median of all angles at non-hyperbolic vertices, effectively eliminating the currently sharpest half of features.

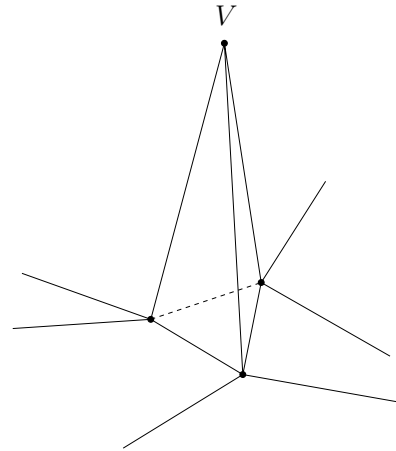


Figure 8. Sharp feature.

Once the vertex for decimation has been identified, its removal requires a re-triangulation of the hole that its

removal leaves. Our scheme to do this is actually an edge-collapse, details of which are described later.

Here then is pseudo-code for our simplification algorithm given an input mesh  $M$ :

**Step 1:** For each non-hyperbolic vertex  $V$  of  $M$  compute  $angle(V)$ ,  $ht(V)$  and  $vol(V)$ .

**Step 2:** Find the median  $A$  of all the  $angle(V)$  values.

**Step 3:** From amongst the  $V$  such that  $angle(V) > A$  find the one with the smallest value of  $vol(V)$ .

**Step 4:** Decimate  $V$  by deleting it and its adjacent edges from  $M$ .

**Step 5:** Re-triangulate the hole in  $M$  resulting from the previous step by applying the routine described below.

**Step 6:** If the desired resolution has been reached, then exit; if not, for each neighbor  $W$  of the most recently deleted vertex  $V$ , check if it is non-hyperbolic in the re-triangulated  $M$ , and, if so, re-compute  $angle(W)$ ,  $ht(W)$  and  $vol(W)$ . Go to Step 2.

#### D. Re-triangulation Routine

Our re-triangulation routine employs a novel but somewhat technical heuristic. Suppose the neighbors of the decimated vertex  $V$  are  $W_1, W_2, \dots, W_k$ . Of the  $O(k^2)$  straight lines through a pair of distinct vertices in  $W_1, W_2, \dots, W_k$ , find the one closest to  $V$ , say the straight line joining  $W_i$  and  $W_j$ . Without loss of generality suppose that  $W_i$  is at least as close to  $V$  as  $W_j$ .

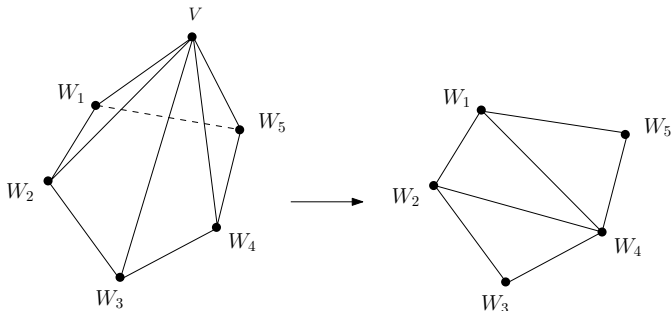


Figure 9. Re-triangulating after decimating  $V$  by collapsing  $VW_4$ .

We shall “collapse” the edge  $VW_i$  by deleting  $V$  and its adjacent edges, and re-triangulate the resulting hole using the fan of triangles with  $W_i$  as the common vertex and every other edge of the link of  $V$  as an opposite edge, except, of course, for the two adjacent to  $W$  (Figure 9 indicates the scheme by collapsing  $VW_4$ ). Motivation for this heuristic is apparent in Figure 10. One can re-triangulate the hole left by decimating  $V$  in that figure in essentially two different ways – by collapsing the edge  $VW_2$  or by collapsing the edge  $VW_1$ . Clearly, the latter choice is more faithful to the original shape and our heuristic finds this.

#### E. Complexity

We omit details in this version but the complexity of the volume cost based method is fairly straightforwardly seen

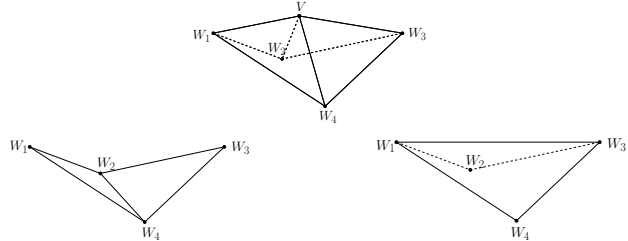


Figure 10. After decimating  $V$  (top figure) the resulting hole can be re-triangulated in two ways: by collapsing  $VW_2$  (lower left) or  $VW_1$  (lower right). Our heuristic chooses the right one which is more faithful to the original surface.

to be linear in the size of the mesh because: (a) the initial classification of vertices in Step 1 requires examination only of each vertex’s link (moreover, each edge of a surface mesh can appear in the link of at most two vertices); (b) Steps 2 and 3 can be executed in a total of linear time with appropriate data structures; (c) Steps 4, 5 and 6 require a total of linear time as well from an amortized cost analysis.

## IV. EXPERIMENTAL RESULTS

We have implemented our volume cost based simplification algorithm using C++ and the STL on an Intel platform with 1.66GHz CPU and 1GB RAM. We used several pre-packaged algorithms from the CGAL library [2] and the Boost C++ library [1]. Parts of our code have been adapted from Jeff Somers [14] simplification routines.

We have so far applied our method to various input meshes of up to about 100K triangles. Output from a fairly small cow mesh was earlier seen in Figure 1. Figures 11 and 12 show much larger models reduced to an identically small fraction of about 1% of the original number of triangles, using both Garland’s quadric-based method and our volume cost based. Figure 13 similarly compares the two again on a smaller mesh. The quality of output of the two methods is clearly comparable.

However, we have yet to make analytic comparisons with help of, for example, the Metro tool [3]. Nor, we have yet made any observation of simplification times on very large meshes (for meshes of size up to 100K our algorithm completes in a few seconds of computation time). We hope to report the results of such experimentation in the near future.

## V. CONCLUSIONS AND FUTURE WORK

We propose a novel and simple mesh simplification algorithm that first classifies vertices as hyperbolic and non-hyperbolic according to their local geometry, and then iteratively decimates vertices after the computation of a volume cost function at non-hyperbolic vertices. Our volume cost based simplification algorithm is easy to implement and initial experiments have produced good output quality.

More comparative experiments and consequent refinement of the algorithm is the next step. We intend to make our software available freely. Theoretical proofs of error bounds are important as well, and may, possibly, be obtained by relating a vertex's estimated volume cost to some real volume values. Another direction for future work is simplification of meshes carrying color and texture data.

#### ACKNOWLEDGMENTS

The software was implemented with help of code from Jeff Somers [14], as well as the CGAL [2] and Boost C++ [1] libraries. We thank Tep Vuthy for early contributions.

#### REFERENCES

[1] Boost C++ Library, url: <http://www.boost.org>.

[2] CGAL 3.2.1 (Computational Geometry Algorithms Library), url: <http://www.cgal.org>.

[3] P. Cignoni and C. Montani and R. Scopigno, *A comparison of mesh simplification algorithms*, Journal of Computers and Graphics, Vol. 22, No. 1, 1998, 37-54.

[4] M. Garland, *Multiresolution modeling: survey & future opportunities*, Eurographics 99: Proceedings of Eurographics 99, 1999, 28-35.

[5] M. Garland and P. S. Heckbert, *Surface simplification using quadric error metrics*, SIGGRAPH '97: Proceedings of the 24th annual conference on Computer graphics and interactive techniques, ACM, 1997, 209-216.

[6] P. S. Heckbert and M. Garland, *Optimal triangulation and quadric-based surface simplification*, Journal of Comput. Geom. Theory Appl., Vol. 14, No. 1-3, 1999, 49-65.

[7] H. Hoppe, *Progressive meshes*, SIGGRAPH '96: Proceedings of the 23rd annual conference on Computer graphics and interactive techniques, ACM, 1996, 99-108.

[8] D. P. Luebke, *A Survey of Polygonal Simplification Algorithms*, University of North Carolina at Chapel Hill, 1997.

[9] D. P. Luebke, *A Developer's Survey of Polygonal Simplification Algorithms*, Journal of IEEE Comput. Graph. Appl., Vol. 21, No. 3, 2001, 24-35.

[10] D. P. Luebke and C. Erikson, *View-dependent simplification of arbitrary polygonal environments*, SIGGRAPH 97: Proceedings of the 24th annual conference on Computer graphics and interactive techniques, ACM, 1997, 194-208.

[11] A. Van Oosterom and J. Strackee, *The Solid Angle of a Plane Triangle*, IEEE Transactions on Biomedical Engineering, Vol. 30, 2005, 125-126.

[12] J. Rossignac and P. Borrel, *Multi-resolution 3D approximations for rendering complex scenes*, Journal of Modeling in Computer Graphics., 1993, 455-465.

[13] W. J. Schroeder and J. A. Zarge and W. E. Lorensen, *Decimation of triangle meshes*, SIGGRAPH '92: Proceedings of the 19th annual conference on Computer graphics and interactive techniques, ACM, 1992, 65-70.

[14] J. Somers, Mesh Simplification Viewer, url: [http://www.jsomers.com/vipm\\_demo/meshsimp.html](http://www.jsomers.com/vipm_demo/meshsimp.html).

[15] J. O. Talton, *A short survey of mesh simplification algorithms*, University of Illinois at Urbana-Champaign, 2004.

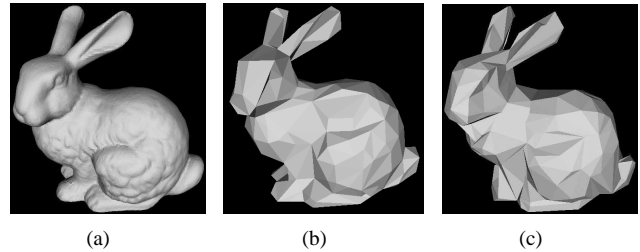


Figure 11. Bunny: (a) Base mesh (69451 triangles) (b) Garland's quadric-based simplification to 500 triangles (c) Volume cost based simplification to 500 triangles.

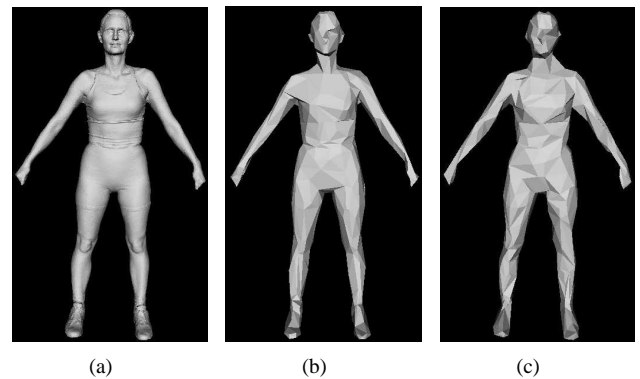


Figure 12. Woman: (a) Base mesh (65994 triangles) (b) Garland's quadric-based simplification to 1000 triangles (c) Volume cost based simplification to 1000 triangles.

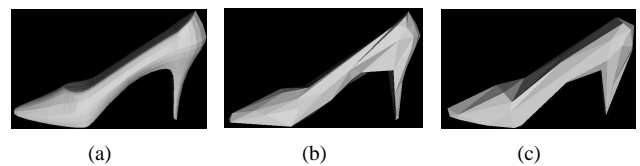


Figure 13. Pump: (a) Base mesh (2268 triangles) (b) Garland's quadric-based simplification to 139 triangles (c) Volume cost based simplification to 139 triangles.